CrowdLang: A Programming Language for the Systematic Exploration of Human Computation Systems

Minder, Patrick ; Bernstein, Abraham

Abstract: Human computation systems are often the result of extensive lengthy trial-and-error refinements. What we lack is an approach to systematically engineer solutions based on past successful patterns.In this paper we present the CrowdLang1 programming framework for engineering complex computation systems incorporating large crowds of networked humans and machines with a library of known interaction patterns. We evaluate CrowdLang by programming a German-to-English translation program incorporating machine translation and a monolingual crowd. The evaluation shows that CrowdLang is able to simply explore a large design space of possible problem-solving programs with the simple variation of the used abstractions. In an experiment involving 1918 different human actors, we show that the resulting translation program significantly outperforms a pure machine translation in terms of adequacy and fluency whilst translating more than 30 pages per hour and approximates the human-translated gold standard to 75%.

CrowdLang: A Programming Language for the Systematic Exploration of Human Computation Systems

Patrick Minder and Abraham Bernstein

Dynamic and Distributed Information Systems Group
University of Zurich, Switzerland
{lastname}@ifi.uzh.ch

Abstract. Human computation systems are often the result of extensive lengthy trial-and-error refinements. What we lack is an approach to systematically engineer solutions based on past successful patterns.
In this paper we present the *CrowdLang*[1] programming framework for engineering complex computation systems incorporating large crowds of networked humans and machines with a library of known interaction patterns. We evaluate *CrowdLang* by programming a German-to-English translation program incorporating machine translation and a monolingual crowd. The evaluation shows that *CrowdLang* is able to simply explore a large design space of possible problem-solving programs with the simple variation of the used abstractions. In an experiment involving 1918 different human actors, we show that the resulting translation program significantly outperforms a pure machine translation in terms of adequacy and fluency whilst translating more than 30 pages per hour and approximates the human-translated gold standard to 75%.

Keywords: *CrowdLang*, Programming Language, Human Computation, Collective Intelligence, Crowdsourcing, Translation Software

1 Introduction

Much of the prosperity gained by the industrialization of the economy in the 18th century was the result of increased productivity after dividing work into smaller tasks performed by more specialized workers. Wikipedia, Google, and other stunning success stories show that with the rapid growth of the World Wide Web and the advancements in communication technology, this concept of Division of Labor can also be applied to knowledge work [1, 2]. These new modes of collaboration—whether they are called collective intelligence, human computation, crowdsourcing, or social computing[2]—are now able to routinely

[1] This work was supported in part by the Swiss National Science Foundation (SNSF-Project: 200021-143411/1). A short research note summarizing a part of the evaluations in this paper was published at the ACM WebSci Conference 2012 [12]

[2] A clear distinction between these concepts is an ongoing debate in the community [3, 4, 2]. Relying on [4] this paper considers *human computation* as computation that is carried out by humans and *human computation systems* as "paradigms for utilizing human processing power to solve problems that computers cannot yet solve"

solve problems that would have been unthinkably difficult only a few years ago by interweaving the creativity and cognitive capabilities of networked humans and the efficiency and scalability of networked machines in processing large amounts of data [5]. The advent of crowdsourcing markets such as Amazon's Mechanical Turk (MTurk) further fosters this development. Hence, Bernstein et al. suggest that we can view these systems as constituting a kind of a "global brain" [5].

Even though a plethora of human computation systems (HCS) exists, our understanding of how to *"program"* these systems is still poor: human computers are profoundly different from traditional computers due to the huge motivational, error and cognitive diversity within and between humans [5]. Hence, HCSs are mostly used for parallel information processing (e.g., image labeling). These tasks share in common that they are massively (or embarrassingly) parallelizable, have a low interdependence between single assignments, and use relatively little cognitive effort. Many tasks, however, cannot be captured in this paradigm. Consider, e.g., the joint editing of lengthy texts as accomplished on Wikipedia. Here, a large number of actors work on highly interdependent tasks that would be very difficult to cast into a bulk parallelization with low interdependence. Hence, to harness the full potential of HCSs, we need powerful new programming metaphors and infrastructures that support the design, implementation, and execution of human computation. Specifically, we need a programming language that supports the whole range of possible dependencies between single tasks, allows for the seamless reuse of known human computation patterns incorporating both humans and machines to exploit prior experience, and integrates multiple possible execution platforms (e.g., micro-task markets, games with a purpose) to leverage a large ecosystem of participants. To move from a culture of *"Wizard of Oz"* techniques, in which applications are the result of extensive trial-and-error refinements, a programming language has to support the recombination [6] of interaction patterns to systematically explore the design space of possible solutions. Recent research only partially addresses these challenges by providing programming frameworks and models [7–9] for massive parallel human computation, concepts for planning and controlling dependencies [10, 11], and theoretical deductive analysis of emergent collective intelligence [2].

In this article, we present the *CrowdLang* human computation programming language and framework for interweaving networked humans and computers. *CrowdLang* supports cross-platform workforce integration, the management of human computer latency, and incorporates abstractions for group decisions, contests, and collaborative interaction patterns as proposed by Malone et al. [2]. *CrowdLang* also supports the management of arbitrary dependencies among tasks and workers, and not only asynchronous parallelization. We show *CrowdLang*'s feasibility and strength by programming a collection of text translation programs. The resulting translation programs are able to speedily translate non-trivial texts from German to English achieving a significantly better quality than pure machine translation approaches. Also, given the simple recombination of patterns supported by *CrowdLang*, we were able to unearth a novel human com-

putation pattern, which we call *"Staged-Contest with Pruning,"* that outperforms all other known patterns in the translation task.

2 Background and Related Work

Relevant to this paper is research about frameworks for supporting the design of HCS and the analysis of emergent collective intelligence.

A number of programming frameworks and concepts addressing the distinct challenges in engineering human computation systems have been proposed recently. Little et al. [7, 13] proposed the use of the imperative programming framework *TurKit*. Investigating workflows composed by iterative and parallel traditional programming constructs, they explored basic technical problems caused by the high latency associated with waiting for a response from a human worker when writing and debugging human computation code. They support the idea of a "crash-and-rerun" programming model, which allows a programmer to repeatedly rerun and debug processes without republishing costly previously completed human computation.

Several programming frameworks inspired by the MapReduce [14] programming metaphor have been proposed to coordinate arbitrary dependencies between interdependent tasks. These frameworks model complex problems as a sequence of partitioning, mapping, and reducing subtasks. For example, Kittur et al.'s *CrowdForge* programming framework [8] starts by breaking down large problems into discrete subtasks either by using human or machine computers. Then human or machine agents are used to collect a set of solutions. Finally, the results of multiple workers are merged and aggregated into the solution of the larger problem. Similarly, Ahmad et al.'s *Jabberwocky* programming environment [9] extended this idea by providing an additional human and resource management system for integrating workforces from different markets, as well as a high-level procedural programming language. Finally, Noronha et al. [15] suggest a "divide-and-conquer" management framework inspired by corporate hierarchies.

These studies highlight the importance of designing new environments for programming human computation systems but are restricted by the structural and synchronous rigidness of the MapReduce programming metaphor when modeling workflows with arbitrary dependencies [16]. Further, they do not provide any explicit treatment of cognitive diversity in and between human actors [5] or abstractions for complex coordination patterns such as group decision procedures [2]. Finally, they assume that computation can be fully specified ex-ante. In many complex problem-solving tasks, however, processes are difficult to specify ex-ante and only gain more specific definitions during execution or may start out as well-defined tasks and then lose their specific definition due to some unexpected exceptions. Thus, it was proposed that processes move along a *specificity frontier* from well defined and static to loosely defined and dynamic [10]. Zhang et al. [11], for example, propose a system that exploits a self-organizing crowd to solve a planning under constraints problem. This system illustrates the crowd-

based solution of a process somewhere in the middle of the specificity frontier. To harness the full potential of human computation systems, we believe that programming languages designed for this purpose should exhibit all these features.

Complementing these (empirical) explorations of possible patterns, several studies [3, 4] taxonomize various aspects of HCSs. Malone et al. [2] examined about 250 different HCSs and identified in the *Collective Intelligence Genome* the characteristics *("genes")* that can be recombined to the basic building blocks *("genome")* of human computation systems. Their conceptual classification framework suggests characterizing each building block by answering two pairs of questions. First, they considered staffing *(Who is performing the task?)* and different kinds of incentives *(Why are they doing it?)*. Second, they analyzed a specific system by defining the goal of a task *(What is being done?)* and problem-solving process *(How is it being done?)*. We believe that this framework is not only suitable for analyzing existing applications but also for designing new ones by recombining the basic building blocks as Bernstein et al. [6] also proposed in the context of business processes.

3 The *CrowdLang* Programming Framework

Conventional programming languages are developed to interoperate with deterministic machines. When moving from programming pure machine computation to hybrid machine-human or pure human computation systems, these languages are not a good match as they lack abstraction for dealing with the cognitive, error, and motivational diversity within and between humans [5] and the varying degrees of detail in many human task definitions.

The objective of *CrowdLang* is to build a general-purpose programming language and framework for interweaving human and machine computation within complex problem solving. *CrowdLang* intends to incorporate explicit methods for handling (cognitive, error, and motivational) diversity, complex coordination mechanisms (and not only batch processing) and abstractions for human computation tasks such as group decision processes. In a future version, the framework will also support the specificity frontier to allow unstructured, constraint-restricted computation and for run-time task decomposition as well as the modeling of non-functional constraints such as budget, completion time, or quality. Last but not least, the *CrowdLang* engine has to address the technical challenges associated with crowd worker latency (waiting for human response) [7].

The framework consists of three major components: *(1)* The *CrowdLang Library* simplifies the design of new human computation systems. It supports the seamless reuse of existing interaction patterns by providing an extensible programming library. The integrated *intelligent discovery assistant* supports the exploration of the whole design space through simple pattern recombination. *(2)* The *CrowdLang Engine* addresses the technical challenges of executing human computation algorithms by managing the crowd latency (waiting for human response), debugging human computation code, and the re-executing of human

computation after exceptions. The *CrowdLang Integrator* integrates different execution platforms such as micro-task markets and games with a purpose.

4 The *CrowdLang* Programming Language

In accordance with Malone et al.'s empirical exploration [2], *CrowdLang* supports operators for task decomposition and group decision processes.

4.1 Basic Operators, Task Decomposition and Aggregation

CrowdLang provides language constructs for defining basic operators, data items, and control flow constructs (see Figure 1). A *Task* represents the transformation of a given problem statement into a solution. The transformation is performed either by humans *(Human Computation)* or machines *(Machine Computation)*. A *Problem Statement* defines a task in terms of a question and the required input data. A *Solution* represents the computed results for a *Problem Statement*. A *Sequence Flow* defines the execution order of single tasks and manages therefore classical producer/consumer relationships, where the produced output of a previous task is consumed as an input in the next task.

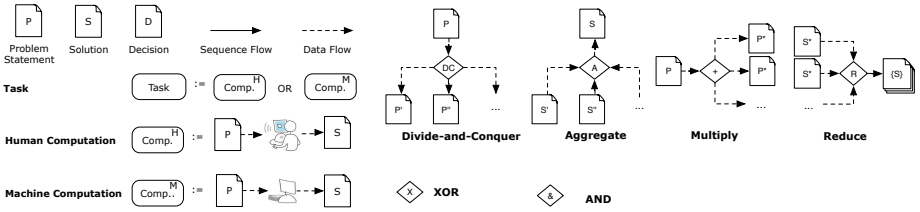


Fig. 1. Basic *CrowdLang* Operators, Routing, Aggregation, and Task Decomposition

CrowdLang provides a set of routing operators to distribute computation and aggregate results (see Figure 1). The *Divide-and-Conquer* operator decomposes a problem statement into multiple parallelizable, distinct subproblems. The *Aggregate* operator, in contrast, aggregates the results of several subtasks to a solution of the initial problem statement.

A given problem can be distributed to actors in three different ways. The *Multiply* control flow operator indicates that a given problem gets transformed multiple times in parallel. Hence, copies of the original problem statement get allocated to multiple independent actors potentially leading to different solutions (in particular when performed by human actors). Hence, the result of such an execution is a set of solutions. The *Reduce* operator takes a set of solutions and determines the "best" solution candidate employing a decision procedure. Together, the *Multiply* and *Reduce* are the building block for many parallelizing crowd computing patterns. *CrowdLang* provides the established *exclusive (XOR)* and *parallel (AND)* control flow operators. XOR is used to create or synchronize alternative paths; AND can be used to create and synchronize parallel paths.

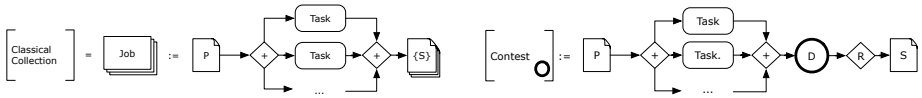
4.2 Building Blocks of Collective Intelligence

In accordance to [2], *CrowdLang* defines a set of basic building blocks classified as *Create* and *Decide* interaction patterns.

Create Interaction Patterns The framework defines two variations of the create interaction pattern: Collection and Collaboration.

A *Collection* occurs when actors independently contribute to a task. Malone et al. [2] illustrated the Collection in terms of posting videos on YouTube. In *CrowdLang* a Collection is defined as a multiplied independent transformation of a problem statement into a proposed solution using the *Multiply* operator. *CrowdLang* defines two variations of the Collection gene. First, A *Job* (see Figure 2a) is a simple *Multiply-AND* combination resulting in a set of solutions.

A *Contest* (see Figure 2b) is a *Job* followed by a *Reduce* selecting the *Job*'s best solution based on a decision.



(a) (b)

Fig. 2. (a) Classical Collection and (b) Contest

A *Collaboration* occurs when actors cooperate either by contributing iteratively or by solving different parts of a problem. *CrowdLang* supports two variations of the collaboration gene (see Figure 3a). First, an *Iterative Collaboration* models problem solving as an *iterative process of interdependent solution improvement* whereas the submitted contributions are strongly interdependent on previous ones. It can be likened to the `repeat ... until <condition>` construct of a typical programming language. A typical example of this process is article writing in Wikipedia, iterative labeling, or OCR. Based on a problem statement, a crowd worker builds an initial version of the solution followed by a decision process where either the crowd or a machine decide whether the proposed solution needs further refinement. This procedure will be repeated until the decision procedure accepts the solution.

Second, a *Parallelized Interdependent Subproblem Solving* represents the combination between a divide-and-conquer of the initial problem, the parallel execution of the partial problems, and the aggregation of the results to the final solution. The main advantage of this pattern is that it makes it possible to first split a problem into a set of independent subproblems that can then be solved in parallel. Open-source programming is an example of this pattern, where an overall problem specification is divided into subsystems, each of which are programmed and then linked together to build the resulting system.

Decide Interaction Patterns A *Group Decision* is defined as a mechanism that determines the best solution by using multiple crowd workers in an inde-

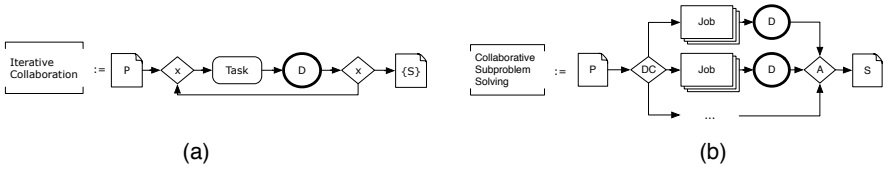


Fig. 3. (a) Collaboration and (b) Parallelized Interdependent Subproblem Solving

pendent manner. Examples of group decisions are the evaluation of different solutions by voting, forced agreement, or parallel guessing with aggregation. An *Individual Decision* is a decision that is the result of an evaluation by a single human or machine agent. Note that these specifications depart from Malone et al.'s framework, under which a group decision is defined as a decision that a group makes that subsequently holds for all participants (e.g., elections, ballot questions for new laws, etc.). Our operationalization allows for group-based decisions that affect only individual actors or affect all individuals differently. This can be exemplified by the use of a recommendation system to aid movie selection.

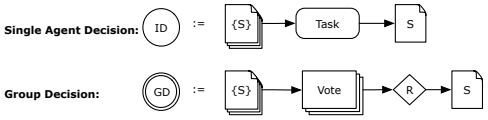


Fig. 4. The Decide Gene: Single Agent and Group Decisions

5 Design a new Application with *CrowdLang*

Using *CrowdLang*, we developed a family of 9 non-trivial text translation programs incorporating human crowd worker and machine translation.

5.1 Translating Text with *CrowdLang*

The development process— incorporating the *CrowdLang Library* and *Intelligent Discovery Assistant (IDA)* for recombining different workflow refinements — included the following five steps:

1. *Identify the Core Activities:* A programmer starts with the definition of an abstract problem-solving algorithm by identifying abstract core activities (operations) and Producer-Consumer dependencies [16] among them.
2. *Define the Design Space:* Then, (s)he selects a set of suitable interaction patterns from the *CrowdLang Library* that can be applied as operators for the abstract core activities.
3. *Generate the Recombinations:* Then the *IDA* systematically generates a set of alternative refinements by recombining the selected patterns.

4. *Execution:* The programmer executes the alternative refinements.
5. *Evaluation:* Finally, (s)he evaluates the generated variations and selects the best algorithm among the set of alternative refinements.

1. Identify the Core Activities We started by defining an abstract problem-solving workflow for the translation task and modeled the core activities and producer-consumer dependencies among them in Figure 5. This workflow starts by first iteratively splitting the input—an article—into paragraphs and then sentences (`Task Decomposition`); then processes the resulting sentences in parallel by sequentially applying machine translation (`MT`) and crowd-based rewriting (`Rewrite`) ; Then, using an aggregate operator (`A`), the sentences are combined into paragraphs that are then assigned to crowd workers to improve the language quality by enhancing paragraph transitions and enforcing a consistent wording (`Improve Language Quality`). Finally, the grammatical correctness is improved (`Check Syntax`) by eliminating syntactical and grammatical errors.

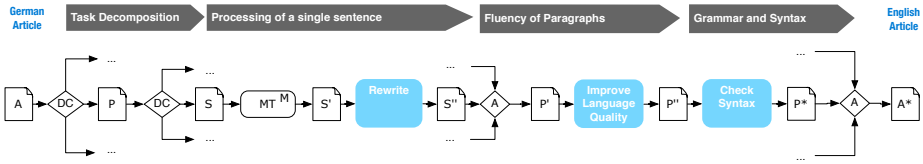


Fig. 5. Abstract translation algorithm

2. Define the Design Space Then, we selected the following set of suitable interaction patterns for the abstract core activities identified in the previous step.

Contest with Six Sigma Pruning (CP) uses an adapted contest pattern to generate semantically correct sentences and improve text quality (see Figure 2). First, 3 different workers generate solutions. Then, these proposed solutions are pruned using the Six Sigma rule [17, p. 320 - 330]. The Six Sigma rule—a method originally used in operations research—intends to improve the output quality of a process by minimizing variability. Specifically, we compared the crowd workers' working time on a task compared to a previously collected average. Defining the average work time as $\widehat{w}$ we hypothesize that tasks should be accomplished within the interval $\widehat{w} \pm 3\sigma$ with σ as the standard deviation of the normal distribution. We minimize the number of "lazy turkers" (someone who tries to maximize his earnings by cheating) by rejecting results of workers when the working time is shorter than the lower bound $\widehat{w} - 3\sigma$. We also eliminate so-called "eager beavers" (people who are going beyond the task requirements) with the upper bound $\widehat{w} + 3\sigma$. We select the best solution among this remaining using a group decision. In particular, we ask 5 workers to rank the proposed solutions and then apply the Borda rule [18] to determine the winning solution.

Iterative Improvement (II) uses a iterative collaboration interaction pattern to generate semantically correct sentences and improve text quality. We define three termination conditions: (1) two out of three crowd workers assess a sentence as semantically correct, (2) the result of an iteration step is equivalent to the previous one, or (3) we exceed the number of three iterations.

Iterative Dual Pathway Structure (DP) is an adaptation of [19]. We assign the same problem (e.g., an initial translation) to two different paths. In each of the two paths, a worker is asked to improve the translation $Comp^{H1}$ and $Comp^{H2}$. At the end of this step the solutions of the two paths are compared. If the two solutions are equivalent based on an individual decision by a third crowd worker then we have a final result. If not, we iterate by sending each of the results back along its path for additional improvements until they are judged equivalent.

Find-Fix-Verify (FFV) [20] checks the grammatical and syntactical correctness of a text fragment by first asking crowd worker to find misspellings and grammatical errors. Then a group of crowd workers is asked to propose a solution for the identified problems. Finally, the solutions are verified by three independent crowd workers. Additionally, we adapted this pattern slightly by also introducing also a spell-checking software $Comp^{M}$.

3. Generate the Recombinations We systematically generated a set of 9 alternative refinements for the algorithm by recombining the previously selected interaction patterns (see Table 1). For each refinement we chose 3 patterns for both `Rewrite` and `Improve Language Quality`.

	CPxCP	CPxII	CPxDD	IIxCP	IIxII	IIxDP	DPxCP	DPxII	DPxDP
Rewrite	CP	CP	CP	II	II	II	DP	DP	DP
Improve L. Quality	CP	II	DP	CO	II	DP	CP	II	DP
Check Syntax	FFV	FFV	FFV	FFV	FFV	FFV	FFV	FFV	FFV

Table 1. Resulting pattern recombinations

5.2 Evaluation

We evaluated the different translation algorithms implemented with *CrowdLang* along a number of dimensions. We compare the results of the 9 runs as well as a pure machine translation with a gold standard human translation using an automatic text analysis measure. The best two program combinations additionally get compared to the gold standard by the crowd as well as professional translators.

Experimental Setup The evaluation was conducted on a standard German to English translation task. Specifically, we generated translations for 15 different articles from Project Syndicate[3]— a Web source of op-ed commentaries —totaling

[3] http://www.project-syndicate.org/

153 paragraphs with 558 sentences and 10'814 words translated from German to English. As a baseline, we considered Google Translate.

Evaluation Aspects First, we considered different performance metrics such as average work time, throughput time (including waiting time), and cost per sentence. Second, based on literature research [21], we judged the translation quality along three different dimensions:

1. *Adequacy:* The meaning of the reference translation is also conveyed by the output of a translation algorithm
2. *Fluency:* The translation being evaluated is judged according to how fluent it is without comparing it against a reference translation.
3. *Grammar:* A translation segment is being evaluated according to its grammatical correctness without comparing it against a reference translation.

Evaluation Methodology The crowd-based translation processes were evaluated using automatic machine, non-professional, and professional human evaluation. First, we approximated the translation quality with the METEOR [22] score, which automatically estimates human judgment of quality using unigram matching between a candidate and reference translation. We considered one reference translation for each evaluation segment. Hence, a translation attains a score of 1 if it is identical to the reference translation.
Second, the translated text went through three stages of human evaluation. A monolingual group consisting of 89 native and 194 non-native speakers of English recruited on MTurk judged a set of 3 randomly extracted sentences with respect to adequacy on an ordinal scale from 1 (None) to 5 (All Meaning) [21]. A monolingual group consisting of 283 participants (140 native and 143 non-native speakers), was asked to judge a randomly extracted sentence with respect to fluency on an ordinal scale from 1 (Incomprehensible) to 5 (Flawless English) [21]. Finally, a bilingual group of 8 professional translators from the Swiss company 24translate (https://www.24translate.ch/) evaluated the translations by comparing each version of a translation to the German source text.

5.3 Results

Automatic Evaluation First, we compared the resulting quality of all 8 recombinations against the baseline. In direct comparison, two algorithms—CPxCP and CPxII—outperformed the baseline (0.29) by reaching a METEOR score of 0.38 and 0.36 respectively as shown in Table 2. Note that we view the awful performance of most other recombinations not as a failure of our approach but as a desired result of a systematic exploration of the design space. Just like in biologic gene recombination, many possible solutions are not viable. Nonetheless, an approach that explores all combinations (or if computationally infeasible most combinations using some optimization approach) is more likely to uncover good solutions such as the CPxCP algorithm than one that tries to apply some kind of heuristic to immediately hone in on good ones.

	CPxCP	CPxII	CPxDD	IIxCP	IIxII	IIxDP	DPxCP	DPxII	DPxDP
METEOR	0.389	0.369	0.335	0.290	0.290	0.290	0.309	0.298	0.285
Precision	0.76	0.74	0.72	0.68	0.68	0.68	0.70	0.68	0.69
Recall	0.71	0.68	0.65	0.64	0.64	0.64	0.64	0.63	0.65

Table 2. Summary of METEOR evaluation

Quantitative Human Evaluation 283 human non-professional evaluators rated the crowd-based translations in respect to adequacy and fluency on average as 3.16 and 3.37 on the ordinal scale from 1 (Incomprehensible) to 5 (Flawless English). In comparison, the professional reference translation scored on average 4.24 and 3.58. As such, the crowd-based algorithms outperform the baseline machine translation and are outperformed by the reference translation. All differences are significant at the 95% level using the non-parametric Friedman test [23]. Furthermore, the 8 professional translators evaluated CPxCP as the best of all non-professional translation algorithms (see Figure 6).

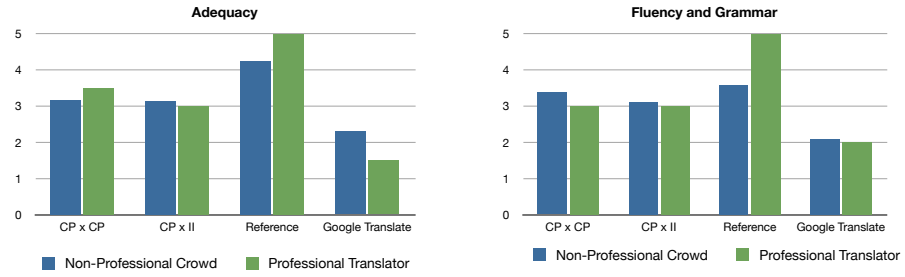


Fig. 6. Mean evaluation scores for the evaluation of adequacy, fluency and grammar by 283 English native speakers and 8 professional translators

Qualitative Evaluation While these results show that the resulting translations are far from perfect, they still make useful translations available at a fraction of the time and cost of traditional solutions. In particular, the analysis of the follow-up interviews with the professional translators and an in-depth analysis of the adequacy, fluency and grammar score distribution (see Figure 7)[4] show that the differences in quality are mostly caused by a few challenges in the German language morphology. For example, Translator-1 judges one of the translations as a *"Good solid translation that reflects exactly what the original says,"* whereas the pure machine translation failed, which was expressed by Translator-2 *"Nonsensical. [...]"* However, in some cases the CPxCP algorithm failed totally.

Using the professional translators' reviews, we were able to identify several types of problem that had occurred in our experiments:

1. Word order and punctuation often lead to problems when the word order provided by the machine translation reflects the morphology of the German

[4] The question as to whether the Likert scale should be considered equi-distant or ordinal is under debate in the social sciences. Here, we interconnected the data points, for illustration purposes only without trying to take a stance on this issue.

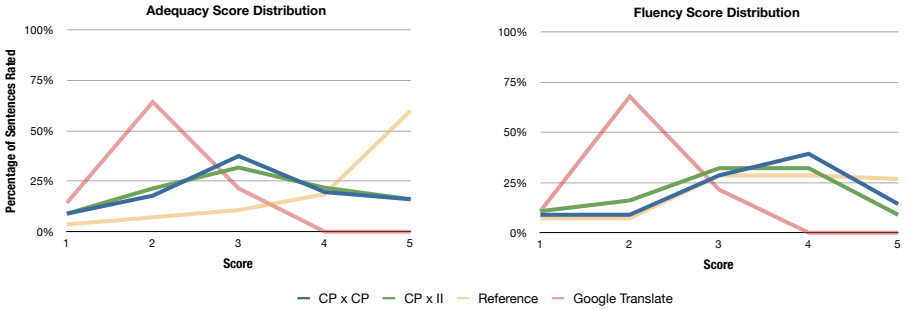


Fig. 7. Proportional score distribution per paragraph for the different translation programs in regard to adequacy and fluency.

language. Translator-5 elucidates this in detail: *"[...] reflects the German original [...]Adverbs come after the verb 'to be' in English. [...]"*

2. Some translations struggle in using appropriate tenses, as expressed by Translator-1 *"This would be fine except for two places that incorrectly use a relative clause with 'which' [...] A reader could still understand it [...]"*
3. In very few instances, problems where observed that should only occur when non-native speakers or machines are editing a translation.

We subsequently found that installing text-improvement "subroutines" in the program to address these specific challenges can significantly improve the results while still keeping the throughput time and costs low. An empirical evaluation of these subroutines is forthcoming.

On average, an article translation was completed within 24 minutes for CPxCP or 35 minutes for CPxII. In terms of cost, the translation of a sentence cost 0.09$ with CPxCP and 0.12$ with CPxII.

6 Discussion, Findings, and Limitations

Our evaluation highlights a number of interesting findings.
(1) The translation programs illustrate that *CrowdLang* lends itself to the simple exploration of a large design space of possible program alternatives. Whilst we cannot provide empirical proof that this feature generalizes to a large number of other applications, it does, however, indicate that a systematic exploration of the design space of possible human computation programs based on known and novel patterns may help to find good solutions. This technique promises to help the transition from an era of *"Wizard of Oz* techniques," where well-functioning programs are the result of lengthy trial-and-error processes, to a more engineering-oriented era - a goal first postulated by Bernstein et al. [20].
(2) The empirical evaluation shows that it is indeed possible to significantly improve the quality of generated translations employing monolingual crowd workers at astonishing speeds. Whilst the translations are far from perfect, they make useful translations available at a fraction of the time and cost of traditional

solutions. We are confident that the incorporation of further text improvement "subroutines" in the program— such as the use of bilingual crowd workers for the most complex German sentence structures only —can solve these kind of problems.
(3) Our adaptation of the Six Sigma rule to human computation allows us to run the processes without any sophisticated pruning techniques. We could forgo any use of "control questions"– a considerable saving in terms of effort. On the downside, our evaluation is limited in that the usage of such quality control measures may have led to better results. An evaluation of this question is forthcoming.
(4) Our pairing of the systematic exploration of the design space with the empirical evaluation helped us to find the novel human computation pattern *Staged Contest with Pruning* (CPxCP). This best-performing pattern combined contests over several stages by pruning the intermediate results using the Six Sigma rule and automatic comparison with the input to uncover cut-and-pastes.
A major limitation is that our programs have so far only been evaluated in German to English translation tasks. An evaluation using standard machine translation tasks (e.g., EU Parliament dataset), exploring the sensitivity of our programs to different machine translation tools, and other language pairs is forthcoming.

7 Conclusion and Future Work

In this paper we introduced *CrowdLang* – a general-purpose framework and programming language for interweaving human and machine computation. Using the practical task of text translation, we illustrated that *CrowdLang* allows the "programming" of complex human computation tasks that entail non-trivial dependencies and the systematic exploration of the design space of possible solutions via the recombination of known human computation patterns.
Our empirical evaluation showed that some of the resulting programs generate "good" translations indicating that the combination of human and machine translation could provide a fruitful area of human computation. Finally, it unearthed a novel human computation pattern: the "Staged Contest with pruning."
We hope that *CrowdLang* will be used by others to implement their human computation programs, as it will allow them to easily compare different solutions.

References

1. Malone, T., Laubacher, R., Johns, T.: General management: The age of hyperspecialization. Harvard Business Review **89**(7-8) (2011) 56–65
2. Malone, T., Laubacher, R., Dellarocas, C.: The collective intelligence genome. MIT Sloan Management Review **51**(3) (2010) 21–31
3. Quinn, A., Bederson, B.: Human computation: a survey and taxonomy of a growing field. In: Proceedings of the 2011 annual conference on Human factors in computing systems, ACM (2011) 1403–1412
4. Law, E., Ahn, L.: Human computation. Synthesis Lectures on Artificial Intelligence and Machine Learning **5**(3) (2011) 1–121

5. Bernstein, A., Klein, M., Malone, T.: Programming the global brain. Communications of the ACM **55**(5) (2012) 1–4
6. Bernstein, A., Klein, M., Malone, T.: The process recombinator: a tool for generating new business process ideas. In: Proceedings of the 20th international conference on Information Systems, Association for Information Systems (1999) 178–192
7. Little, G., Chilton, L., Goldman, M., Miller, R.: Turkit: human computation algorithms on mechanical turk. In: Proceedings of the 23nd annual ACM symposium on User interface software and technology, ACM (2010) 57–66
8. Kittur, A., Smus, B., Khamkar, S., Kraut, R.: Crowdforge: Crowdsourcing complex work. In: Proceedings of the 24th annual ACM symposium on User interface software and technology, ACM (2011) 43–52
9. Ahmad, S., Battle, A., Malkani, Z., Kamvar, S.: The jabberwocky programming environment for structured social computing. In: Proceedings of the 24th annual ACM symposium on User interface software and technology, ACM (2011) 53–64
10. Bernstein, A.: How can cooperative work tools support dynamic group process? bridging the specificity frontier. In: Proceedings of the 2000 ACM conference on Computer supported cooperative work, ACM (2000) 279–288
11. Zhang, H., Law, E., Miller, R., Gajos, K., Parkes, D., Horvitz, E.: Human computation tasks with global constraints, CHI (2012)
12. Minder, P., Bernstein, A.: How to translate a book within an hour - towards general purpose programmable human computers with crowdlang. In: ACM Web Science 2012, New York, NY, USA (2012)
13. Little, G., Chilton, L., Goldman, M., Miller, R.: Exploring iterative and parallel human computation processes. In: Proceedings of the ACM SIGKDD workshop on human computation, ACM (2010) 68–76
14. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. Communications of the ACM **51**(1) (2008) 107–113
15. Noronha, J., Hysen, E., Zhang, H., Gajos, K.: Platemate: crowdsourcing nutritional analysis from food photographs. In: Proc. of the 24th annual ACM symposium on User interface software and technology, ACM (2011) 1–12
16. Malone, T., Crowston, K.: The interdisciplinary study of coordination. ACM Computing Surveys (CSUR) **26**(1) (1994) 87–119
17. Chase, R., Aquilano, N., Jacobs, F.: Operations management for competitive advantage. McGraw-Hill/Irwin New York (2006)
18. Young, H.: An axiomatization of borda's rule. Journal of Economic Theory **9**(1) (1974) 43–52
19. Chen, Y., Liem, B., Zhang, H.: An iterative dual pathway structure for speech-to-text transcription. In: Human Computation: Papers from the AAAI Workshop (WS-11-11). San Francisco, CA, August. (2011)
20. Bernstein, M., Little, G., Miller, R., Hartmann, B., Ackerman, M., Karger, D., Crowell, D., Panovich, K.: Soylent: a word processor with a crowd inside. In: Proceedings of the 23nd annual ACM symposium on User interface software and technology, ACM (2010) 313–322
21. Papineni, K., Roukos, S., Ward, T., Zhu, W.: Bleu: a method for automatic evaluation of machine translation (2002)
22. Banerjee, S., Lavie, A.: Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization (2005) 65
23. Iman, R., Davenport, J.: Approximations of the critical region of the friedman statistic. Technical report, Sandia Labs., Albuquerque, NM (USA); Texas Tech Univ., Lubbock (USA) (1979)